1. Introduction

The C4I For the Warrior (C4IFTW) vision has been stated as follows:

> *The Warrior needs a fused, real-time, true-picture of the battlespace and the ability to order, respond, and coordinate vertically and horizontally to the degree necessary to prosecute the mission in that battlespace.*

This broad visionary statement demonstrates that an unprecedented degree of integration and interoperability is required of DOD systems, both for legacy systems and for systems that are under construction. The Defense Information Infrastructure (DII) Common Operating Environment (COE) is the key to achieving this vision.

The DII COE[1] originated with a simple observation about command and control systems: certain functions (mapping, track management, communication interfaces, etc.) are so fundamental that they are required for virtually every command and control system. Yet these functions are built over and over again in incompatible ways even when the requirements are the same, or vary only slightly, between systems. If these common functions could be extracted, implemented as a set of extensible low-level building blocks, and made readily available to system designers, development schedules could be accelerated and substantial savings could be achieved through software reuse. Moreover, interoperability would be significantly improved because common software is used across systems for common functions, and the functional capability only needs to be built correctly once rather than over and over again for each project.

This observation led to the development of the DII COE. Although its roots are in the C4I arena, the DII COE and its principles are not unique to C4I. The DII COE has been expanded to encompass a range of other functional areas including logistics, transportation, base support, personnel, health affairs, and finance. All new DISA systems are being built using the DII COE while existing DISA systems are being migrated to use the DII COE. OSD has recently issued a directive[2] that all new military systems, except for weapon control systems, shall use the DII COE.

Three representative DISA systems that use the DII COE are the Global Command and Control System (GCCS), the Global Combat Support System (GCSS), and the Electronic Commerce Processing Node (ECPN) system. All three systems use the same infrastructure and integration approach, and the same COE components for functions that are common between the systems.

GCCS is a C4I system that began with two main objectives: replacement of the World-Wide Military Command and Control System (WWMCCS) and implementation of the C4I For the Warrior concept (C4IFTW). Functionally, as a C4I For the Warrior system, GCCS

[1] The acronyms "DII COE" and "COE" are used interchangeably throughout this document. Other COEs exist (such as the JMCIS COE) which are very similar in scope or implementation with the DII COE. To avoid confusion, unless otherwise indicated, "COE" always refers to the DISA DII COE.

[2] OSD Directive dated 30 August 1996 (Subject: Implementation of the DOD Joint Technical Architecture).

includes multiple workstations cooperating in a distributed LAN/WAN environment. Key features include "push/pull" data exchange, data processing, sensor fusion, dynamic situation display, analysis and briefing support, and maintenance of a common tactical picture among distributed GCCS sites. GCCS is already fielded at a number of operational CINCs. In calendar year 1996 GCCS completed the first objective by replacing all WWMCCS systems. Attention is now being devoted to enhancing GCCS to more completely realize the C4IFTW concept. An important step in this evolution is to integrate GCCS and GCSS in a seamless way to provide the warrior with C4I capabilities and reachback to the CONUS sustaining base infrastructure.

GCSS is under development and is targeted for the warfighting support functions (logistics, transportation, etc.) to provide a system that is fully interoperable with the warfighter C4I system. Implemented to its fullest potential, GCSS will provide both warfighter support (including reachback from deployed commanders into the CONUS sustaining base infrastructure) and cross-functional integration on a single workstation platform.

ECPN is also under development and is to provide the foundation for paper-less exchange of business information, including funds transfer, using electronic media. ECPN is important in the history of the DII COE because it represents a radically different application than either GCCS or GCSS. This demonstrates the fact that the DII COE principles are mission-domain independent.

The DII COE is described more completely in later chapters of this document as is technical information required to properly access and extend software contained within the COE. The concepts herein represent the culmination of open systems evolutionary development from both industry and government with contributions from each of the services and several agencies. The resulting COE architecture is an innovative framework for designing and building military systems. Because it reuses software contributed by service/agency programs, it utilizes field-proven software for common functions. The engineering procedures for adding new capabilities and integrating systems are mature and have been used for several production releases. The end result is a strategy for fielding systems with increased interoperability, reduced development time, increased operational capability, minimized technical obsolescence, minimal training requirements, and minimized life-cycle costs.

1.1 A Brief History of the DII COE

Initial DII COE development was driven by a near-term requirement to build a suitable WWMCCS replacement. WWMCCS maintenance costs were significant and the system had reached the point of technical obsolescence. A significant aspect of the COE challenge, which was successfully met for WWMCCS with GCCS, is to strategically position the architecture so as to be able to take advantage of technological advances. At the same time, the system must not sacrifice quality, stability, or functionality already in the hands of the warrior. In keeping with current DOD trends, the COE emphasizes use of commercial products and standards where applicable to leverage investments made by commercial industry.

To achieve the near-term WWMCCS replacement objective, technical experts and program managers from each of the services, the intelligence community, the Defense Mapping Agency (DMA), and other interested agencies met for several months beginning in the fall of 1993. Participants proposed candidate systems as a possible starting point for a COE architecture or as a suitable candidate for providing capabilities to meet WWMCCS replacement requirements. None of the candidate systems met all requirements, but it was clear that a combination of the "best" from several systems could produce a near-term system that would be suitable for WWMCCS replacement. Moreover, an infrastructure could be put into place and a migration strategy defined to preserve legacy systems until migration to the intended architecture could be realized.

The cornerstone architectural concept jointly developed during that series of meetings was the GCCS COE. This initial COE was limited in scope to address the immediate C4I problem (i.e., WWMCCS replacement), but its principles, structure, and foundation deliberately went far beyond just the C4I mission domain. The GCCS COE was composed of software contributed from several candidate systems evaluated by this original joint engineering team.

An initial proof-of-concept system, GCCS 1.0, was created and installed in early 1994 at one operational site to validate the approach and to receive early feedback. GCCS 1.1 followed in the summer of 1994 and was the first attempt to integrate software from the Army AWIS and Navy JMCIS programs as initial GCCS COE components. GCCS 1.1 included mission applications from a variety of other programs operating in a "federated" mode. That is, the mission applications were integrated together so as to be able to run on the same hardware without interfering with each other, but not yet able to effectively share data between applications. This successful effort allowed GCCS 1.1 to be installed and tested at beta sites and was used at certain operational sites to monitor events during the 1994 Haiti crisis. GCCS 2.0 fielding began in early 1995 at a number of operational sites. GCCS 2.1 was fielded in mid-1995 and by mid-1996 had successfully replaced WWMCCS. A prototype version of GCCS 2.2 was the basis for JWID 95 and a refinement of it was the basis for JWID 96. The rapid prototyping capability of the GCCS COE allowed another GCCS 2.2 enhancement to be placed in theater to support Bosnia operations and for contingency planning when tensions in the Gulf area increased in mid-1996.

Starting in mid-1995, because GCCS was well underway, technical experts again met under DISA guidance to expand the GCCS COE into the DII COE. The result is a COE that contains all of the original GCCS COE functionality and that is backwards compatible. But the DII COE is was expanded to include global data management and workflow management for GCSS logistics applications and to address other mission domains. Much of the original software has been updated to take advantage of further technological advances and Commercial Off-the-Shelf (COTS) software has replaced some of the original Government Off-the-Shelf (GOTS) components. From this historical perspective, the GCCS COE can be viewed as a proper subset of the much larger DII COE.

Development of the present version of the *I&RTS* and DII COE began in early 1996. It builds upon its predecessor by adding new capabilities to address Web-based applications and to recognize the increasing importance of database issues in building interoperable systems. The Shared Data Environment (SHADE) is a major new addition to the COE to improve interoperability at the data level, to allow data sharing, and to reduce data integration conflicts.

Although GCCS succeeded in replacing the aged WWMCCS, it is important to realize that GCCS is far more than just a WWMCCS replacement. The GCCS system:

- provided more capability than users had before,
- modernized the human/computer interface to simplify training,
- greatly increased the "jointness" of the system,
- increased the suite of tools warriors could use in executing their mission, and
- met many of the requirements for generating a fused view of the Joint Battlespace.

Perhaps more importantly, GCCS is a milestone that demonstrates that the DII COE concept is crucial in being able to rapidly integrate software from candidate programs to successfully build a baseline with an ever-increasing level of functionality. And the DII COE provides the mechanism for rapidly placing these capabilities into the hands of the warrior. GCCS is presently being migrated to the new DII COE 3.0.

The DII COE has its roots in command and control, but the principles and implementation described in this document are not unique to GCCS or GCSS. The principles and implementation are not limited to command and control or logistics applications but are readily applicable to many other application areas. The specific software components selected for inclusion in the COE determine the mission areas that the COE can address. Work is ongoing to refine and optimize the components in the COE and to further populate the COE with software required for Electronic Commerce/Electronic Data Interchange (EC/EDI), transportation, base support, personnel, health affairs, and finance applications. Work is also ongoing to take advantage of new technology advances, such as Web and CORBA, which provide new technology infusion to ensure system longevity. Efforts in data and security are ongoing to complete the goal of true interoperability while not introducing undue security risks. Backwards compatibility is a fundamental tenet of the COE and significant effort is expended to preserve legacy investments. Systems which

migrate to the DII COE now are protected by the backwards compatibility guarantee as future COE versions are released so that upgrading from one COE version to the next is less difficult than most commercial counterparts.

1.2 The DII COE Concept

The DII COE concept is a fundamentally new approach that is much broader in scope than simple software reuse. Software reuse itself is not a new idea. Unfortunately, most software reuse approaches to date have been less than satisfactory. Reuse approaches have generally emphasized the development of a large software repository from which designers may pick and choose modules or elect to rebuild modules from scratch. It is not sufficient to have a large repository, and too much freedom of choice leads to interoperability problems and duplication of effort. This rapidly negates the advantages of software reuse.

Software reuse strategies have also ignored the importance of data reuse. The approach has traditionally been to encapsulate data into a relational database from which applications may retrieve the data according to their own view (i.e., schema). While this approach was a tremendous advance, it fell short of the goal of providing truly interoperable systems in the joint arena. What is required is an approach that promotes data sharing within systems and between systems. The approach must also recognize and resolve the issues of duplicative data, inconsistencies in the data, and data replication. SHADE is the data reuse strategy for the DII COE.

The DII COE emphasizes both software reuse and data reuse and interoperability for both data and software. But its principles are more far reaching and innovative. The COE concept encompasses:

- an architecture and approach for building interoperable systems,
- an environment for sharing data between applications and systems,
- an infrastructure for supporting mission-area applications,
- a rigorous definition of the runtime execution environment,
- a reference implementation on which systems can be built,
- a collection of reusable software components and data,
- a rigorous set of requirements for achieving DII[3] compliance,
- an automated toolset for enforcing COE principles and measuring DII compliance,
- an automated process for software integration,
- an approach and methodology for software and data reuse,
- a set of Application Program Interfaces (APIs) for accessing COE components, and
- an electronic process for submitting/retrieving software and data to/from the DII repository.

This document is first and foremost a Computer Science document that describes *how* modules must interact in the target system. System architects and software developers retain considerable freedom in building the system, but runtime environmental conflicts and data conflicts are identified and resolved through automated tools that enforce COE

[3] The term "DII compliance" is preferred instead of "COE compliance" and is used throughout the *I&RTS*. The compliance concept and approach has not changed, but compliance is measured for segments within the COE as well as mission-application segments that lie outside the COE. Therefore, "DII compliance" is more descriptive and correct than "COE compliance."

principles. An important side effect is that traditional integration tasks largely become the responsibility of the developer. Developers are required to integrate and test their software within the COE prior to delivering it to the government. This simplifies integration because those who best understand the software design (the original developers) perform it, it reduces the cost because integration is performed earlier and at a lower level in the process, and it allows the government to concentrate on validation instead of integration.

In the context of this document, the COE must be understood as a multi-faceted concept. Proper understanding of how the many facets interact is important in appreciating the scope and power of the DII COE and in avoiding confusion in understanding COE material. The next subsection deals with four specific facets in more detail:

- the COE as a system foundation,
- the COE as an architecture,
- the COE as a reference implementation, and
- the COE as an implementation strategy.

Failure to understand these facets will lead to confusion and non-compliant systems.

1. To view the COE as a C4I system is incorrect because it misses the fundamental point that the COE is *not* a system; it is a *foundation* for building an open system. This viewpoint makes fielding and update schedules confusing because it fails to account for the impact of the evolutionary development strategy.

2. To view the COE as GCCS or just an architecture for C4I systems gives the mistaken impression that its principles are limited to the GCCS program or to command and control applications. GCCS is simply the first system build on top of the DII COE.

3. To view the COE as just a recommendation fails to account for the fact that a baseline reference implementation already exists. The baseline is composed of components selected from mature service/agency programs. Such a limiting view leads to the erroneous assumption that the COE is merely a set of guidelines when in fact the DII COE is *both* a set of standards and specifications, *and* a set of pre-built components that implement those standards and specifications.

4. Finally, to view the COE as just an implementation strategy is a limited perspective because it fails to account for the fact that its near-term development is driven by real-world objectives and schedule priorities to meet service/agency commitments. Long-term COE strategy is driven by technology and mission requirements, but near-term fielding schedules ensure that immature technology is not introduced before it is ready. However, short-sighted decisions that may preclude taking advantage of new technology advances are avoided. This limited view also ignores the evolutionary nature of the COE and mission-applications development and it ignores the derived requirement to provide an easy update mechanism for operational sites.

1.2.1 The DII COE as a System Foundation

Figure 1-1 is a greatly simplified diagram that shows how the DII COE serves as a foundation for building multiple systems. Details such as specific COE components, databases, and the internal structure of the COE have been omitted for clarity. Chapter 2 provides this level of information and describes the COE in much more detail. The purpose of Figure 1-1 is just to introduce the concept.

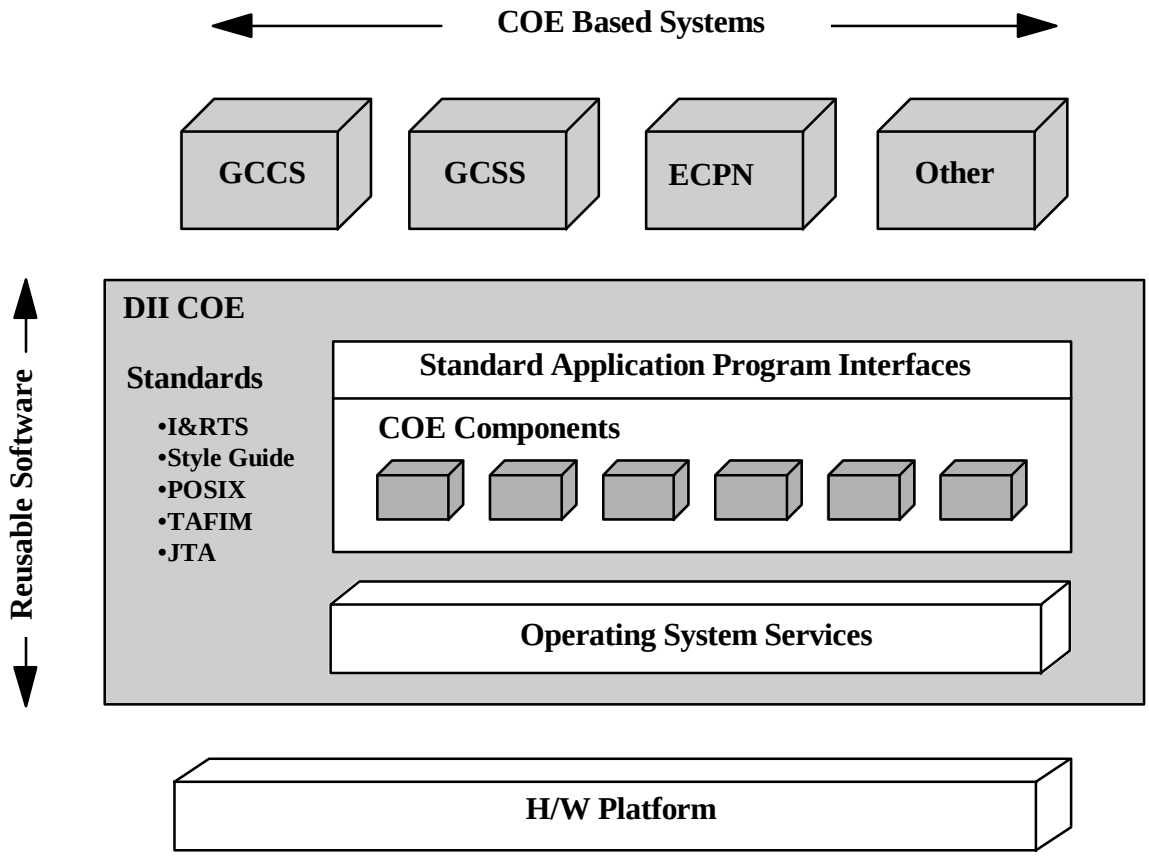

Figure 1-1: DII COE and COE-Based Systems

The shaded box in Figure 1-1 shows two types of reusable software: the operating system and COE components. For the present discussion, it is sufficient to note that COE components are accessed through APIs and that the COE components form the architectural backbone of the target system. The API is the means through which a system permits a programmer to develop applications through interaction with the underlying COE. A set of standards (*POSIX* in the diagram) and specifications (*TAFIM*, *JTA*, *I&RTS*, and *Style Guide* in the diagram) dictate how COE components are to be built and how external components must be built to be compliant with the COE architecture.

Building a target system, such as GCCS or GCSS, is largely a matter of combining COE components with mission-specific software. The COE infrastructure manages the flow of data through the system, both internally and externally. Mission-specific software is mostly concerned with requesting data from the COE and then presenting it in a form that is most meaningful to the operator (e.g., as a pie chart, in tabular form, as a graph). The COE provides the necessary primitives for such data manipulation and has the necessary information about where the requested data is stored, whether locally or remotely across the LAN/WAN. This frees the system designer to concentrate on meaningful data presentation and not on the mechanics of data manipulation, network communications, database storage, etc.

It must be kept in mind, however, that there is only one COE regardless of the target system. The COE is a set of building blocks. System designers select those building blocks required for their mission application, while excluding building blocks that are not required. Each derived system uses the same set of APIs to access common COE components, the same approach to integration, and the same set of tools for enforcing COE principles. For common functions (e.g., communications interfaces, dataflow management), each target system uses precisely the same COE software components from the reference implementation. Compliant systems do not implement their own versions of algorithms within the COE because this will rapidly lead to interoperability problems as algorithms are interpreted differently or because systems fail to upgrade algorithms at the same time. This approach to software reuse significantly reduces interoperability problems because if the same software is used, it is not possible to have two systems that interpret or implement standards differently.

The next subsection describes the features of a few, selected systems. They are presented as examples of COE-based systems to demonstrate the flexibility of the COE, and to demonstrate the potential for a much higher degree of interoperability between systems than ever before.

1.2.1.1 GCCS as a COE-Based System

GCCS is a system specifically designed to meet C4I requirements of the warrior at various echelons within the command structure. It consists of geographically distributed workstations inter-connected via LAN/WAN technologies on a classified network called the SIPRNET (Secret Internet Protocol Router Network). The features provided and the LAN/WAN topology allow warriors to collaboratively share mission responsibilities. Collaborative planning is possible in areas as diverse as creating Time-Phased Force and Deployment Data (TPFDD), distributing Air Tasking Orders (ATO), performing intelligence analysis, and maintaining a common view of the battlefield with up-to-date display of the deployment of all joint and enemy forces.

The GCCS system provides a suite of capabilities across a number of areas that include the following:

- Manpower Requirements Analysis
- Force Planning
- Collaborative Mission Planning
- All Source Data Fusion & Correlation
- Office Automation
- Logistics Support
- Status of Readiness Reports
- Cartographic and Imagery Display and Analysis
- Transportation Planning
- Resource Management
- Fuel Resource Planning
- Teleconferencing
- Scheduling and Movement
- Medical Planning
- Intelligence Analysis
- Communications and Message Handling

The sheer magnitude and capability of GCCS can quickly overwhelm even the most experienced operators. However, the COE provides system administration tools to allow site administrators to selectively install only those software applications required for the site. This minimizes hardware requirements and simplifies site administration. Site administrators can further tailor the installation so that operators are given access to only those applications that pertain to their mission area or for which they have the proper clearances. GCCS allows an operator to access any authorized function from any workstation so that privileges are tied to the operator, not a specific workstation.

Software updates are periodically made available as new capabilities are developed, or as software patches are created to fix problems. Site administrators can receive these updates via tapes, or electronically across the SIPRNET. Electronic updates are available in either a “push” mode (i.e., the update process is initiated electronically by a DISA Software Support Activity) or a “pull” mode (i.e., the update process is initiated electronically by the operational site).

1.2.1.2 GCSS as a COE-Based System

GCSS is designed to fulfill warfighter acquisition and logistics support functions. As with GCCS, the system consists of geographically distributed workstations inter-connected via LAN/WAN technologies. While GCCS is on a classified network (SIPRNET), GCSS will be primarily on an unclassified network (NIPRNET) with direct connectivity to the Internet. Operators have shared access to technical manuals, drawings, Engineering Change Proposals (ECPs), and status of work in progress regardless of their geographic location. This collaborative feature of GCSS is similar to the teleconferencing capability of GCCS and is supported by the same COE infrastructure. The GCSS system effectively integrates people and organizations, data and information, and work processes across the enterprise.

GCSS provides a comprehensive suite of capabilities related to the acquisition process, and to logistics support. Many of the capabilities, such as office automation, are identical to GCCS and hence use the same COE components. Other requirements, such as the need to support the CALS standard, are unique to GCSS. Major features include the following:

- Engineering Drawings Support
- Depot Maintenance Support
- Materiel Management
- Technical Orders
- Workflow Management and Metrics
- Pert Charts
- Logistics Support Analysis
- Access to Non-destructive Imaging Data
- Training Plans
- Reliability Data Management
- Configuration Management
- Teleconferencing
- Office Automation
- CALS Support
- Cost and Schedule Tracking

GCSS uses the same COE system administration tools as GCCS to allow site administrators to selectively install only those software applications required for the site. This minimizes hardware requirements and simplifies site administration. Site administrators can further tailor the installation so that operators are given access to only those applications that pertain to their area of responsibility.

Software updates are periodically made available as new capabilities are developed, or as software patches are created to fix problems. Site administrators can receive these updates via tapes, or electronically across the NIPRNET.

1.2.1.3 ECPN as a COE-Based System

The EC/EDI goal is to automate the procurement process from the point a Request for Quotation is issued by the government, through contract award, through issuance of Purchase Orders, and through actual billing and payment. This requires integration of Accounting, Receiving, Manufacturing, Purchasing, Shipping, and Sales departments in a seamless information backbone. Presently under development, ECPN is the first step in the overall strategy. ECPN is a far reaching modernization effort that greatly simplifies the procurement process through electronic exchange of business data.

ECPN shares several requirements, such as office automation, with both GCCS and GCSS. Moreover, it is being built with the same architectural infrastructure, development methodology, site administration, and basic building blocks for communications and data exchange as GCCS and GCSS. Many of the requirements, such as specific message formats, are unique to the EC/EDI commercial world.

ECPN is important in the evolution of the DII COE for several reasons.

- It requires interfacing with commercial systems and vendors as well as government systems and personnel.
- It requires guaranteed delivery and acknowledgment of receipt of critical data (e.g., financial transactions).
- It requires a different view of security as compared to traditional DOD usage. Emphasis is on assurance of confidentiality of data, authenticity of the data, and authenticity of the source of the data rather than sensitivity to classification of the data.

- It requires handling enormous data rates nation-wide. The data rates are easily in the millions of transactions per day when fully implemented because every federal financial transaction must be accounted for.
- It requires serious consideration of denial of service. Downtime of the system can have enormous financial and legal ramifications.
- It requires consideration of a wide range of target platforms. ECPN processing nodes can be specified because they are under government control at selected megacenters, but the target hardware for commercial vendors cannot be predicted *a priori*. The mix is a cross-section of hardware/software from the computer industry.

The challenges in building ECPN are sure to refine the DII COE in terms of the components required to populate the COE and in the principles and techniques which define the COE.

1.2.2 The DII COE as an Architecture

The DII COE is a "plug and play" open architecture designed around a client/server model. Functionality is easily added to or removed from the target system in small manageable units called *segments*. Segments are defined in terms of functions that are meaningful to operators, not in terms of internal software structure. Structuring the software into segments in this manner is a powerful concept that allows considerable flexibility in configuring the system to meet specific mission needs or to minimize hardware requirements for an operational site. Site personnel perform field updates by replacing affected segments through use of a simple, consistent, graphically-oriented user interface.

The DII COE model is analogous to the Microsoft Windows® paradigm. The idea is to provide a standard environment, a set of standard off-the-shelf components, and a set of programming standards that describe how to add new functionality to the environment. The Windows paradigm is one of "federation of systems" in that properly designed applications can coexist and operate in the same environment. But simple coexistence is not enough. It must be possible for applications to share data. The DII COE extends the Windows paradigm to allow for true "integration of systems" in that mission applications share data at the server level.

Federation versus integration is an important architectural distinction. However, integration is not possible without strict standards that describe how to properly build components to add to the system. This applies equally to software functions and data. This document and other related documents detail the technical requirements for a well-behaved, DII-compliant application. The COE provides automated tools to measure compliance and to pinpoint problem areas. A useful side effect of the tools and procedures is that software integration is largely an automated process, thus significantly reducing development time while automatically detecting potential integration and runtime problem areas.

More precisely, to a developer the DII COE is:

- **An Architecture:** A precisely defined TAFIM and JTA-compliant, client/server architecture for how system components will interact and fit together and a definition of the system-level interface to COE components.

- **A Runtime Environment:** A standard runtime operating environment that includes "look and feel," operating system, and windowing environment standards. Since no single runtime environment is possible in practice, the COE architecture provides facilities for a developer to extend the environment in such a way as to not conflict with other developers.

- **A Data Environment:** A standard data environment that prescribes the rules whereby applications can share data with other applications.

- **A Reference Implementation:** A clearly defined set of already implemented, reusable functions. A set of reusable software and data is a cornerstone of the DII COE product.

- **A Set of APIs:** A collection of interfaces for accessing COE components. Thus, the COE is a set of building blocks in the same sense that X Windows and Motif are building blocks for creating an application's Graphical User Interface (GUI).

- **A Set of Standards and Specifications:** A set of rules that describe how to use the COE, how to construct segments, how to create a GUI, etc.

- **A Development Methodology:** A process for developing, integrating, and distributing the system and a process for sharing components with other developers. The COE emphasizes and encourages incremental development that has the advantage of quickly producing usable functionality.

DISA maintains the software in an online configuration management repository called SDMS (Software Distribution Management System). This decreases the development cycle by allowing developers to receive software updates and to submit new software segments electronically.

1.2.3 The DII COE as a Reference Implementation

The COE necessarily includes an implementation of the components defined to be in the COE. The reference implementation is the key to reusability and interoperability. Use of the reference implementation provided is required to assure interoperability and is therefore a fundamental requirement for DII compliance. The reference implementation may change over time to take advantage of new technologies or to fix problem reports, but incremental improvements are introduced while preserving backwards compatibility.

1.2.4 The DII COE as an Implementation Strategy

The COE is also an evolutionary acquisition and implementation strategy. This represents a departure from traditional development programs. It emphasizes incremental development and fielding to reduce the time required to put new functionality into the hands of the warrior, while not sacrificing quality nor incurring unreasonable program risk or cost. This approach is sometimes described as a "build a little - test a little - field a lot" philosophy. It is a process of continually evolving a stable baseline to take advantage of new technologies as they mature and to introduce new capabilities. But the changes are done one step at a time so that the warfighters always have a stable baseline product while changes between successive releases are perceived as slight. Evolutionary development has become a practical necessity for many development programs because the traditional development cycle time is longer than the technical obsolescence cycle time. This approach allows program managers the option of taking advantage of recently developed functions to rapidly introduce new capabilities to the field, or of synchronizing with COE development at various points for those situations where incremental upgrades are not readily acceptable to the customer community.

From the perspective of a COE-based system, the recommended implementation strategy is to field new releases at frequent intervals. Each release might include enhancements to both the COE and mission-area applications. Mission-area applications are considered to be provisional, subject to user feedback. Applications for which feedback is favorable are retained in subsequent releases and hardened as needed for continued operational use. As appropriate, mission applications that are widespread in use and commonality will be integrated into the COE or evolved to add new features.

The COE implementation strategy is carefully structured to protect functionality contained in legacy systems so that over time they can migrate to full COE utilization. Legacy systems must use only "public" APIs and migrate away from use of "private" APIs. Public APIs are those interfaces to the COE that will be supported for the life cycle of the COE. Private APIs are those interfaces that are supported for a short period of time to allow legacy systems to migrate from unsanctioned to sanctioned APIs. All new development is required to use only public APIs and use of any other APIs results in a non-DII compliant segment. The process of migrating from existing legacy "stove-pipe" systems to utilize the COE is a primary source for articulating technical requirements for the COE and it provides program managers with information useful to establish development priorities.

From the perspective of a system developer, whether developing a new application or migrating an existing one, the COE is an open client/server architecture that offers a collection of services and already built modules for mission applications. Thus, the developer's task is to assemble and customize existing components from the COE while developing only those unique components that are peculiar to particular mission requirements. In many if not most cases, this amounts to adding new "pull-down menu entries and icons."

1.3 Lessons Learned

The COE as the embodiment of an architectural concept offers the opportunity to leverage a mature, proven, field tested software base for a wide variety of applications for the services, agencies, and Joint community. As budgets shrink and as budgetary priorities shift, program managers require the ability to continue to respond rapidly with systems that satisfy the information needs of United States and Allied Armed Forces. The COE implementation strategy is a significant advancement in fulfilling this ongoing need.

Examination of state-of-the-art development in light of these realities results in a set of fundamental tenets that greatly influence the history, future, and direction of the DII COE. An explanation of these tenets is useful in understanding the COE as a whole.

- *Pre-COE practices lead to development and redevelopment of the same functionality across systems*. Redevelopment is frequently necessary because of technological changes as algorithms are improved or as hardware becomes faster and cheaper. However, development cost tends to be high due to a lack of coordination between programs that share common requirements.

- *Duplication of functionality within the same system is more expensive than avoiding duplication*. Lack of coordination between program developers is a fundamental cause for duplicative functions, but an additional factor is that reuse libraries are not commonly available. The impact is more than just program costs. System users are often given conflicting information even in the presence of identical data because designers took slightly different approaches to solving the same problems or made slightly different assumptions.

- *Interoperability is not achievable through "paper" standards alone*.[4] Standards are necessary, but not sufficient[5] to guarantee interoperability. Interoperability problems are generally caused, not by the standards chosen, but by differing or incorrect interpretations of standards. System designers often choose different standards with which to comply, but even when the standards are the same, different interpretations of the standards can greatly change the way the resulting system operates. The COE emphasizes use of industry and government standards, but relies even more on automated ways of measuring and evaluating compliance, and thus quantitatively evaluating program risk. The only practical way to achieve interoperability is to use exactly the same software, written to appropriate standards, for common functions across applications. For example, the COE contains a common tactical track correlator to ensure that all users see the same tactical picture. The answer produced by the

[4] This statement is not meant to minimize the importance of standards, but to state that they alone are not sufficient to solve interoperability problems. The situation would be far more desperate in the absence of standards.

[5] The solution provided by the COE is to define specifications and a reference implementation of a standard. For example, in the user interface area, Motif is the standard selected for Unix platforms and the *DII Style Guide* is the specification written to be compliant with Motif, but tailored for the particular mission domain.

correlator may be incorrect, but it will be incorrect for all users. But this also means that a problem correction in one place then becomes effective for all users.

- *Pre-COE practices lead to exponential growth in testing and associated development costs*. Lack of commonality and modularity in system building blocks means that there is much duplication of effort in testing basic functionality and testing in one section of a system is often tightly coupled to testing in another section. This complicates and extends the certification process. Configuration management, system integration, and long-term maintenance are also more complex and costly when there is a lack of commonality and modularity in system building blocks.

- *The importance of training is usually underestimated and the magnitude of the training problem is increasing*. An operator is often expected to use multiple systems which behave completely differently, are equally complex with their own subtleties, and which give slightly different answers. Operator turnover is rapidly reaching the point where the time it takes to train an operator is a significant portion of the time that the operator is assigned to his current tour of duty. Training is greatly reduced by a consistent "look and feel" and by the ability to present to the operator only those functions useful for his task.

- *Don't reinvent the wheel*. If a component already exists, it should probably be utilized even if the component is not the optimum solution. Almost any module can be improved but that is rarely the issue. Reuse of existing and proven software allows focus of attention on mission uniqueness. Rather than concentrating scarce development resources on recreating building blocks, the resources can be more appropriately applied to customization and development of functionality that is not already available.

- *Utilize existing commercial standards, specifications, and products whenever feasible*. The commercial marketplace generally moves at a faster pace than the military marketplace and advancements are generally available at a more rapid rate. Use of commercial products has several advantages. Using already built items lowers production costs. The probability of product enhancements is increased because the marketplace is larger. The probability of standardization is increased because a larger customer base drives it.

1.4 Assumptions and Objectives

The following assumptions apply to the DII COE:

- The DII COE will migrate to full compliance with the *JTA* standards profile[6] when it is fully defined. These standards promote an open systems architecture, the benefits of which are assumed to be well known and generally accepted.

- The DII COE is to be hardware independent and will operate on a range of open systems platforms running under standards-based operating systems. Program-driven requirements, associated testing costs, and funding will dictate which specific hardware platforms are given priority.

- Non-developmental items (NDIs), including both COTS and GOTS products, are the preferred implementation approach.

- The DII COE is programming-language neutral. It does not state a preference of one language over another, but leaves the selection of a programming language to higher-level standards profile guidance and programmatic considerations. When a selection is to be made, C++ is recommended over C while Ada95 is recommended over any earlier versions. Any statements in the *I&RTS* which appear to state or imply a preference for one language over another are unintentional.

COE development is driven by C4IFTW requirements as articulated by the services through the appropriate DISA Configuration Control Board (CCB) process. Development priorities are established by the CCB Chair and given to the DII COE Chief Engineer for implementation.

The broad program drivers for the DII COE lead to a number of program objectives that include those stated in the *TAFIM, Volume 2*:

1. **Commonality**: Develop a common core of software that will form the foundation for Joint systems, initially for C4I and logistics systems.

2. **Reusability**: Develop a common core of software that is highly reusable to leverage the investment already made in software development across the services and agencies.

3. **Standardization**: Reduce program development costs through adherence to industry standards. This includes use of commercially available software components whenever possible.

4. **Engineering Base**: Through standardization and an open architecture, establish a large base of trained software/systems engineers.

[6] *JTA* replaces the standards guidance in the *TAFIM* as per OSD directive (Subject: Implementation of the DOD Joint Technical Architecture) dated 30 August 1996.

5. **Training**: Reduce operator training costs and improve operator productivity through enforcement of a uniform human-machine interface, commonality of training documentation, and a consistent "look and feel."

6. **Interoperability**: Increase interoperability through common software and consistent system operation.

7. **Scalability**: Through use of the segment concept and the COE architectural infrastructure, improve system scalability so that COE-based systems will operate with the minimum hardware resources required.

8. **Portability**: Increase portability through use of open systems concepts and standards. This also promotes vendor independence for both hardware and software.

9. **Security**: Improve system security to the extent possible to protect the system from deliberate attack and prevent unauthorized access to data and applications.

10. **Testing**: Reduce testing costs because common software can be tested and validated once and then applied to many applications.

1.5 Document Scope

This document describes the technical requirements for building and integrating software components on top of the DII COE. It provides implementation details that describe, from a software development perspective, the following:

- the Common Operating Environment (COE) approach to software reuse,
- the runtime execution environment,
- the Shared Data Environment (SHADE),
- the requirements for DII compliance,
- how to structure components to automate software integration, and
- how to electronically submit/retrieve software components to/from the software repository.

Note: This document supersedes all earlier draft versions, presentations, or working group notes. It specifically supersedes all previous GCCS or DII Integration documents. All segments submitted to DISA are required to be in accordance with this document.

1.6 Applicable Documents, Standards, and Specifications

This document is one in a series of related documents that define development requirements, system architecture, engineering tools, and implementation techniques. Many of the documents cited are available on the World-Wide-Web, or contact the DISA Configuration Management office for information on how to obtain the desired documents.

Because the COE and COE-based systems are ongoing programs, enhancements and additional features are developed on a regular basis. Documentation updates are regularly released for each of the documents listed here. Be sure to always refer to the latest version for the documents listed below, and be aware that many of the documents are being modified and extended to address DII COE-based systems, not just GCCS or GCSS.

1. *Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)*, **January 1996, DISA Center for Computer Systems Engineering**. This document is the definitive high-level technical description of the COE. It documents the architectural design produced by the DISA COE Design Working Group. It is useful for understanding how the client/server model has been implemented within the DII COE.

2. *C4ISR Architecture Framework*, **CISA-0000-104-96, Version 1.0, 7 June 1996, C4ISR Integration Task Force (ITF) Integrated Architectures Panel**. This document presents an innovative definition of levels of interoperability. The DII COE adopts these levels of interoperability and maps DII compliance to interoperability levels.

3. *Defense Information Infrastructure (DII) Common Operating Environment (COE) Version 3.0 Baseline Specifications*, **31 October 1996, DISA**. This document describes the detailed contents of each COE release and is updated with each subsequent release. It includes the name and version of each segment in the COE as well as COTS products, their version, and applicable patches.

4. *Defense Information Infrastructure (DII) Common Operating Environment (COE) System Requirements Specification*, **Draft, 1996, Institute for Defense Analysis**. Service and Agency requirements for a COE are defined in this document. It is a living document that is updated as necessary to reflect ongoing requirements collection.

5. *Defense Information Infrastructure Software Quality Compliance Plan*, **Draft, 1 January 1996, DISA**. This document describes a plan for evaluating COE segments from a software quality perspective. The plan includes static analysis of segment source code to measure complexity, maintainability, risk, and other standard software metrics.

6. *Department of Defense Joint Technical Architecture, Final Coordination Draft 0.9*, **17 June 1996, Joint Technical Architecture Working Group**. The *JTA* has been mandated by OSD directive for "... all emerging systems and systems upgrades. The

JTA applies to all C4I systems and the interfaces of other key assets (e.g., weapons systems, sensors, office automation systems, etc.) with C4I systems. The *JTA* also applies to C4I Advanced Concept Technology Demonstrations and other activities that lead directly to the fielding of operational C4I capabilities." The *JTA* stipulates DII compliance as part of its requirements. It also "... replaces the standards guidance in the *Technical Architecture Framework for Information (TAFIM)* currently cited in DOD Regulation 5000.2-R."

7. *Department of Defense Technical Architecture Framework for Information Management, Volumes 1-8*, **Version 3.0 (Draft), 30 September 1995, DISA Center for Architecture**. This multi-volume document defines a standards profile and the DOD *Technical Reference Model (TRM)* for information management systems. This document set also presents a high-level technical architecture that is useful for classifying levels within a system's infrastructure. The *TRM* distinguishes between the hardware platform, hardware-specific services, supporting infrastructure services, and mission applications.

8. *Information Technology - Portable Operating System Interface for Computer Environments (POSIX) - Part 1: System Application Program Interface (API) [C Language]*, ISO 9945-1, 1990; *Information Technology - Portable Operating System Interface for Computer Environments (POSIX) - Part 2: Shell and Utilities*, **ISO 9945-2, 1993**. The POSIX documents are an ongoing standardization effort that is attempting to define a common set of low-level functions, especially at the operating system level, across all hardware platforms and operating systems.

9. *User Interface Specification for the Defense Information Infrastructure (DII)*, **Version 2.0, 1 April 1996, DISA**. This document, sometimes called the *DII Style Guide*, defines the "look and feel" of the user interface for COE-based systems. The style guide provides specifications for applications using Motif and Windows GUIs; a future version of the document will include Windows NT and Web-based applications.

1.7 Document Structure

This document is structured to correspond to the typical phases in a development cycle, beginning with how a developer builds a segment, submits it to the government, and then how it is fielded to an operational site. Chapter 1 of this document is an overview of the DII COE, a brief history of its development, and applicable documents and standards.

Chapter 2 gives a brief technical description of the COE, its components, and the principles that determine whether a software component is part of the COE or is a mission application. Selection of the particular components to populate the COE determine what applications can be supported, but the principles which define a COE are not application-specific. Chapter 2 also describes the important concept of DII compliance and maps compliance to levels of interoperability.

Chapter 3 is an overview of the development process. It includes a discussion of the process from segment registration through development, submission to DISA, integration, and site installation. The tools provided in the COE and how they are used is key to understanding automated integration.

Chapter 4 describes SHADE and other database considerations within the context of the COE. Databases are heavily used within COE-based systems, and early consideration of their structure, how they are to be used, and how they are to fit into the overall system is crucial in building a successful system.

Chapter 5 describes the runtime environment as it exists for operational sites, the disk directory and file structure fundamental to the COE, and the procedures for integrating segments into a runtime environment. Requirements detailed in Chapter 5 must be carefully followed so that applications will not interfere with each other, and so that integration is largely an automated process.

Chapters 6, 7, and 8 are new with this version of the *I&RTS*. They describe extensions for the COE reference implementation that runs on NT platforms, extensions to the COE to support Web applications, and support for DCE applications respectively.

Chapter 9 provides some suggestions for setting up a software development environment. Few requirements are stipulated for a development environment, allowing as much freedom for developers and program managers as possible.

Chapter 10 describes two important components for both developers and operational sites: the online COE Software Distribution Management System (SDMS), and the COE Information Server (CINFO). These components are used to disseminate and manage software, documentation, meeting notices, and general information of importance to the COE community.

Appendix A lists the currently supported COE configurations. The appendix includes supported hardware, and supported COTS versions. It also describes the Reference

Implementation program whereby vendors may obtain low-level components of the COE and port them to their hardware platforms.

Appendix B presents a checklist for developers to use as an aid in determining the degree to which a segment is DII-compliant. As described in the appendix, some conditions are mandatory, others require a migration strategy to show conformance, while others are optional but recommended. This appendix has been reworded and reformatted to be clearer and easier to apply, but is otherwise unchanged from the previous *I&RTS* version.

Appendix C describes the automated tools provided with the COE. A number of new tools are provided to simplify the segment development and maintenance life cycle. The philosophy is to provide developers with access to the same tools that integrators will use so that segment integration is performed, as much as possible, by segment developers prior to segment delivery. Integration of segments with the COE is the responsibility of the segment developer. Government integrators serve as validators *only* in this process to ensure that developers produce DII-compliant segments. In addition to segment validation, government integrators perform system-level integration of all segments submitted by all developers to create the target system.

Appendix D gives additional information on the online repository (SDMS) and the information server (CINFO).

Segment registration is required in order to identify potential conflicts as early in the development cycle as possible. Appendix E describes how to register a segment and what information is required for registration.

The remaining appendices provide additional information on products within the COE, such as the RDBMS, that are either vendor-specific or product-version-specific.

Finally, a List of Acronyms used in the *I&RTS* are presented and a Glossary of frequently encountered terms. The acronyms and terms are encountered throughout DII COE-related documents.

This page is intentionally blank.